Communicating Numerical Values between Humans and Computers

- Recall that **all inputs** entered by users from the keyboard is actually an ASCII code

  This also applies when the entry is a **number**.

  For example, if the program prompts the user to enter an **integer value** and the user wants to enter the number 12, then he/she would type the keys '1' and '2', which will cause keyboard to transmit the ASCII codes 00110001 for '1' and 00110010 for '2'

  On the other hand, the **integer value** 12 is represented inside the computer by the **2's complement code** 00001100.

  Therefore, the ASCII codes in the input must first be **transformed** into a 2's complement representation (by a pretty complicated process)

- NOTE: The Java API library has provided the programmers with such conversion program.

  After you read in a line (consisting of ASCII codes) from the keyboard using:

  ```
  BufferedReader stdin = new BufferedReader
                              (new InputStreamReader(System.in));

  String s = stdin.readLine();
  ```

  You can convert this string of ASCII codes into a 2's complement representation with the **parseInt()** library function:

  ```
   int i = Integer.parseInt(s);
  ```

  The following material will basically show you what is going on inside this **parseInt()** library function....

Converting ASCII input number into 2's complement code

- I will use a concrete example to explain the process to make things easier to follow.

  I will use the input string "12" - which actually consists of the two character ASCII codes 00110001 and 00110010

  The output 2's complement representation for the value 12 is ofcourse 00001100

- First, you have to understand the difference between the **character** '1' and the **integer** 1

The representation for the character '1' is 00110001 (binary)

The (8 bit) representation for the integer 1 is 00000001 (binary)

So to obtain the value that is represented by the character '1', we subtract 00110000 from the ASCII code for '1' (00110001):

```
   '1'   ----->  00110001
               - 00110000
            ---------------
                 00000001
```

- Since the character represented by the code 00110000 is '0', we can also write:

```
   '1'   ----->  00110001
               - (int) '0'
            ---------------
                 00000001
```

- Here is a start of the program that is used to convert ASCII number representation to 2's complement representation:

  The input "12" is processed from left to right. When the program processes the first digit '1', it performs the following calculation:

```
value = 0;

value =  (int) '1' - (int) '0';
```

  This will assign the **integer** 1 to value (i.e., value = $00000000000000000000000000000001_{(2)}$);

- When the program processes the second digit '2', it would process the string "12" and must obtain the binary value 00000000000000000000000000001100

  This can achieved by the following statement:

```
value =  10*value + ( (int) '2' - (int) '0' );
            ^               ^
            |               |
            |        This difference produces the integer value 2
            |
        Since value was 1, this multiplication results in 10
```

  The above statement will assign the **integer** 12 to variable value (i.e., value = $00000000000000000000000000001100_{(2)}$);

  If you really must know, the computer performs all the operations in binary:

```
         10 =                                00001010
      value =        00000000000000000000000000000001 *
    ----------------------------------------------------
   10*value =        00000000000000000000000000001010
  '2' - '0' =                                00000010 +
    ----------------------------------------------------
                     00000000000000000000000000001100  ---> represents 12
```

- The program must process **every digit** in the input ASCII number to obtain the final value.
- **Program code:**

```
// parseInt(s): returns 10's complement integer representation for string s

public static int parseInt(String s)
 {
   int startpos, sign;
   int value;
   int i;

   /* -------
      Make sure the string is a positive number
      ------- */
   if (s.charAt(0) == '-')
    { sign = -1;
      startpos = 1;
    }
   else
    { sign = 1;
      startpos = 0;
    }

   /* -------
      Convert string to two's compl. representation

      Eg:   "123"  --->   1
                          1*10 + 2 = 12
                          12*10 + 3 = 123
      ------- */
   value = 0;
   for (i = startpos; i < s.length(); i++)
    {
      value = ((int) s.charAt(i) - (int) '0') + 10*value;
    }

   if (sign == -1)
      value = -value;

   return(value);
 }
```

- **Example Program:** (Demo above code)
  - Prog file: click here

The **parseInt** method has one additional step not discussed, namely: checking for a "minus" symbol. This step is relatively easy....

The following program is a "demo" version of the **parseInt** method that will spit out a lot of intermdiate data to show you what's going on in the process: AtoiDemo.java

---

Converting 2's complement code to ASCII code for printing....

- Note that the terminal is an "ASCII oriented" device, meaning that a terminal "speaks the ASCII language" and you must "talk to it in ASCII"....

  So it you have an **integer variable** and this variable contains the 2's complement code for the value 72 (so it has 00000000000000000000000001001000), and you send the binary representation to the terminal, the terminal will promptly print..... **the character H** !!! (because that's 72 is the ASCII code for H)...

  In order to see "72" printed to the terminal, you would have to send these following ASCII codes to the terminal:

  ```
  00110111 (for '7')  and 00110010 (for '2')
  ```

- Here is a program that prints the **integer** 72: click here

  - Compile it and run it using the command:

    ```
    java Print72 > out
    ```

  - Use the following command to look what was printed:

    ```
    cat out
    ```

  - Then use the following command to look inside the output file:

    ```
    od -x out
    ```

    It will show an "hex dump" of the content of the file. Can you see the ASCII codes ?

- When you write programs in Java and use:

  ```
  System.out.println(....)
  ```

  to print out integers, the function will **first** convert the 2's complement representation into a String of ASCII codes and then send the ASCII codes to the output.

- Converting a 2's compl. encoding to ASCII digit string (for output):

  The process of converting a 2's compl. encoding to ASCII digit string is as follows (I will also use **decimal notation** because you are most comfortable with this notation. All calculations are done in **binary** within the computer).

  I will use a simple example to illustrate:

```
  00000000000000000000000000001100                 |         12
                                                   |
    divide by 00000000000000000000000000001010     |       divide by 10
                                                   |
                                                   |
  Quotient =  00000000000000000000000000000001     |    Quotient  = 1
  Remainder = 00000000000000000000000000000010     |    Remainder = 2
                                                   |
---------------------------------------------------------------------------
Save the remainder and repeat the steps using the quotient (if Q > 0)
---------------------------------------------------------------------------
                                                   |
  00000000000000000000000000000001                 |         1
                                                   |
    divide by 00000000000000000000000000001010     |       divide by 10
                                                   |
                                                   |
  Quotient =  00000000000000000000000000000000     |    Quotient  = 0
  Remainder = 00000000000000000000000000000001     |    Remainder = 1
                                                   |
---------------------------------------------------------------------------
Save the remainder and stop (Q = 0)
---------------------------------------------------------------------------
```

  - The string of remainders forms the number in the "reverse" order.

- **Note:**

  - The remainder obtained by the divisions are in the 2's complement representation
  - It is easy to obtain the corresponding **character ASCII code**: simple add the ASCII code for '0' to the value

- **Program code:**

```
// toString(x): converts 10's complement value x into an ASCII string

public static String toString(int value)
 {
   int sign, i, j;
   String result;
   char next_digit;
   char next_char;

   /* ---------------------------------------------
      Take care of the value 0
      --------------------------------------------- */
   if (value == 0)
      return("0");

   /* ---------------------------------------------
      Make sure the number to convert is positive
      --------------------------------------------- */
   if (value < 0)
    { sign = -1;
      value = -value;
```

```
  }
  else
   { sign = 1;
   }

  /* ----------------------------------------------------------
     Convert number

             N            N/10     N%10
           =================================
                                          --- result ""
     E.g.:   123    --->   12   and  3    --- result "3"
             12     --->   1    and  2    --- result "23"
             1      --->   0    and  1    --- result "123"
     ------------------------------------------------------------ */
  result = "";

  /* --------------------------------------------
     Take care of all other values (except 0)
     -------------------------------------------- */
  while ( value > 0 )
  {
     next_digit = value % 10;       // reminder = next digit

     next_char = (char) (next_digit + '0') ;  // Convert to ASCII code

     result = next_char + result;   // Put digit at start of number

     value = value / 10;            // remove the processed digit

  }

  // Put in the negative sign if needed....
  if (sign == -1)
   {
     result = "-" + result;
   }
  else
   {
     result = "" + result;    // optional...
   }


  return(result);
}
```

- **Example Program:** (Demo above code) Example
    - Prog file: click here

I also have a "Demo version" of the same program that shows how the process works: click here